

# 1 Sustainable Security & Safety: Challenges and 2 Opportunities

3 **Andrew Paverd**

4 Microsoft Research Cambridge  
5 andrew.paverd@ieee.org

6 **Marcus Völp**<sup>1</sup>

7 University of Luxembourg  
8 marcus.voelp@uni.lu

9 **Ferdinand Brasser**<sup>1</sup>

10 TU Darmstadt, Germany  
11 ferdinand.brasser@trust.tu-darmstadt.de

12 **Matthias Schunter**<sup>1</sup>

13 Intel Labs, Germany  
14 matthias.schunter@intel.com

15 **N. Asokan**

16 Aalto University, Finland  
17 asokan@acm.org

18 **Ahmad-Reza Sadeghi**<sup>1</sup>

19 TU Darmstadt, Germany  
20 ahmad.sadeghi@trust.tu-darmstadt.de

21 **Paulo Esteves-Veríssimo**

22 University of Luxembourg  
23 paulo.verissimo@uni.lu

24 **Andreas Steininger**

25 TU Wien, Austria  
26 steininger@ecs.tuwien.ac.at

27 **Thorsten Holz**

28 Ruhr-University Bochum, Germany  
29 thorsten.holz@rub.de

## 30 — Abstract —

---

31 A significant proportion of today's information and communication technology (ICT) systems are  
32 entrusted with high value assets, and our modern society has become increasingly dependent on  
33 these systems operating safely and securely over their anticipated lifetimes. However, we observe a  
34 mismatch between the lifetimes expected from ICT-supported systems (such as autonomous cars)  
35 and the duration for which these systems are able to remain safe and secure, given the spectrum of  
36 threats they face. Whereas most systems today are constructed within the constraints of foreseeable  
37 technology advancements, we argue that long term, i.e., *sustainable security & safety*, requires  
38 anticipating the unforeseeable and preparing systems for threats not known today. In this paper, we  
39 set out our vision for sustainable security & safety. We summarize the main challenges in realizing  
40 this desideratum in real-world systems, and we identify several design principles that could address  
41 these challenges and serve as building blocks for achieving this vision.

42 **2012 ACM Subject Classification** Security and privacy → Systems security; Software and its  
43 engineering → Software reliability

---

<sup>1</sup> corresponding author



44 **Keywords and phrases** sustainability, security, safety

45 **Digital Object Identifier** 10.4230/OASIS.CERTS.2019.4

46 **Supplement Material** a long version of this paper is available at [http://www.icri-cars.org/wp-](http://www.icri-cars.org/wp-content/uploads/2019/01/s3-vision.pdf)  
47 [content/uploads/2019/01/s3-vision.pdf](http://www.icri-cars.org/wp-content/uploads/2019/01/s3-vision.pdf)

48 **Acknowledgements** This work was supported by the Intel Research Institute for Collaborative  
49 Autonomous and Resilient Systems (ICRI-CARS). The authors thank Muhammad Shafique for his  
50 helpful suggestions on this manuscript.

## 51 **1 Introduction**

52 With the exception of a handful of systems, such as the two Voyager spacecraft control  
53 systems and the computers in the US intercontinental ballistic missile silos,<sup>2</sup> information  
54 and communication technology systems (ICT) rarely reach a commercially viable lifetime  
55 that ranges into the 25+ years we have come to expect from long-lived systems like cars<sup>3</sup>,  
56 as shown in Figure 1. Worse, rarely any networked ICT system stays secure over such a  
57 lifetime, even if actively maintained.

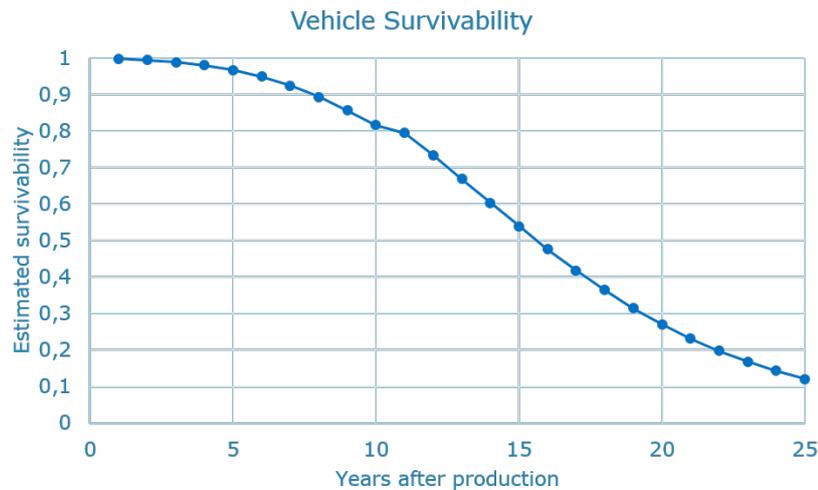
58 Despite this potential risk, current ICT subsystems are already being integrated into  
59 systems with significantly longer design lifetimes. For example, electronic control units in  
60 modern (self-driving) cars, the networked control systems in critical infrastructure (e.g., cyber-  
61 physical systems in power plants and water treatment facilities), and computer controlled  
62 building facilities (a.k.a. smart buildings) are all examples of this mismatch in design lifetimes.  
63 This is also applicable beyond cyber-physical systems: for example, genomic data is privacy-  
64 sensitive for at least the lifetime of the person (if not longer), and yet it is being protected by  
65 cryptographic algorithms with a significantly shorter expected lifespan [7]. For the currently  
66 envisioned limited lifetime, the (functional) safety community has developed techniques to  
67 prevent harm despite accidental faults [10]. However, for the ICT systems in the above  
68 scenarios we aim to preserve security and safetime for the design lifetime, and often beyond.  
69 This means that they need to preserve the value and trust in the assets entrusted to them,  
70 their safety properties (i.e., resilience to accidental faults) *and* their security properties (i.e.,  
71 resilience to malicious faults, such as targeted and persistent attacks). Even when compared  
72 to existing approaches to achieve resilience (such as the safety approaches mentioned above),  
73 these lifetimes translate into ultra-long periods of secure and safe operation.

74 To address this impending challenge, we introduce a new paradigm, which we call  
75 *sustainable security & safety* (S3). The central goal is that a system should be able to  
76 maintain its security and safety, but desirably also its functionality for at least its design  
77 lifetime. This is particularly relevant for systems that have significantly longer design lifetimes  
78 than their ICT subsystems (e.g., modern cars, airplanes, and other cyber physical systems).  
79 In its broadest sense, S3 encompasses both technical and non-technical aspects, and some of  
80 the arising challenges span both aspects.

81 From a technical perspective, S3 brings together aspects of two well-studied technical  
82 fields: *security* and *dependability*. Security research typically begins by anticipating a certain  
83 class of adversary, characterized by a threat model based on well known pre-existing threats

<sup>2</sup> <http://www.dailymail.co.uk/news/article-2614323/Americas-feared-nuclear-missile-facilities-controlled-computers-1960s-floppy-disks.html>

<sup>3</sup> [https://www.aarp.org/money/budgeting-saving/info-05-2009/cars\\_that\\_last.html](https://www.aarp.org/money/budgeting-saving/info-05-2009/cars_that_last.html)



■ **Figure 1** Survival probability of passenger cars by age [2] (based on 1977-2003 NVPP data)

84 at the time the system is designed. From this model, defense mechanisms are then derived for  
 85 protecting the system against the anticipated adversaries or for recovering from these known  
 86 attacks. Dependability, on the other hand, begins with the premise that some components  
 87 of a system will fail, and investigates how to tolerate faults originating from these known  
 88 subsystem failures. Again, the type, likelihood, and consequences of faults are assumed to be  
 89 known and captured in the fault and fault-propagation models.

90 The term *security-dependability gap* [11] is a prototypical example of why established  
 91 solutions from the dependability field cannot be directly used to address security hazards,  
 92 and vice-versa. This arises from the way risks are assessed in safety-critical systems: safety  
 93 certifications assess risks as a combination of stochastic events, whereas security risks arise as a  
 94 consequence of intentional malicious adversaries, and thus cannot be accurately characterized  
 95 in the same way. What is a residual uncertainty in the former, becomes a strong likelihood  
 96 in the latter.

97 Therefore, there are two defining characteristics of S3:

- 98 ■ Firstly, it aims to *bridge the safety-security gap* by considering the complementarity of  
 99 these two fields as well as the interplay between them, and addressing the above-mentioned  
 100 problems under a common body of knowledge, seeking to prevent, detect, remove and/or  
 101 tolerate both accidental faults and vulnerabilities, and malicious attacks and intrusions.
- 102 ■ Secondly, it aims to *protect systems beyond the foreseeable horizon* of technological (and  
 103 non-technological) advances and therefore cannot be based solely on the characterizations  
 104 of adversaries and faults as we know them today. Instead we begin from the premise  
 105 that a long-lived system will face changes, attacks, and accidental faults that were not  
 106 possible to anticipate during the design of the system.

107 The central challenge is therefore to design systems that can maintain their security and  
 108 safety properties in light of these unknowns.

## 109 2 System Model

110 Sustainable security & safety is a desirable property of any critical system, but is particularly  
 111 relevant to long-lived systems, especially those that are easily accessible to attackers. These  
 112 systems are likely to be comparatively complex, consisting of multiple subsystems and

## 4:4 Sustainable Security & Safety: Challenges and Opportunities

113 depending on various external systems. Without loss of generality, we use the following  
114 terminology in this paper, which is aligned with other proposed taxonomies, such as [4]:

- 115 ■ **System:** a composition of multiple subsystems. The subsystems can be homogeneous  
116 (e.g., nodes in a swarm) or heterogeneous (e.g., components in an autonomous vehicle).
- 117 ■ **Failure:** degradation of functionality and/or performance beyond a specified threshold.
- 118 ■ **Error:** deviation from the correct service state of a system. Errors may lead to subsequent  
119 failures.
- 120 ■ **Fault:** adjudged or hypothesized cause of an error (e.g., a dormant vulnerability of the  
121 system through which adversaries may infiltrate the system, causing an error which leads  
122 to a system failure).
- 123 ■ **System failure:** failure of the overall system (typically with catastrophic effect).
- 124 ■ **Subsystem failure:** failure of an individual subsystem. The overall system may be  
125 equipped with precautions to tolerate certain subsystem failures. In this case, subsystem  
126 failures can be considered as faults in the overall system.
- 127 ■ **Single point of failure:** any subsystem whose failure alone results in system failure.

128 S3 aims to achieve the following two goals:

- 129 1. **Avoid system failure:** The primary goal is to avoid failure of the overall system,  
130 including those originating from unforeseeable causes and future attacks. Note that  
131 avoiding system failure refers to both the safety and security properties of the system  
132 (e.g., a breach of data confidentiality or integrity could be a system failure, for certain  
133 types of systems).
- 134 2. **Minimize overheads and costs:** Anticipating and mitigating additional threats  
135 generally increases the overheads (e.g., performance and energy) as well as the initial  
136 costs (e.g., direct costs and increased design time) of the system.

137 In addition, higher system complexity (e.g., larger code size) may lead to increased fault  
138 rates and an increased attack surface. On the other hand, premature system failure may  
139 also have associated costs, such as downtime, maintenance costs, or penalties. Therefore,  
140 a secondary goal of S3 is to develop technologies and approaches that minimize all these  
141 potential overheads and costs.

### 142 **3 Challenges of Long-Term Operation**

143 In our S3 paradigm, we accept that we cannot know the precise nature of the attacks, faults,  
144 and changes that a long-lived system will face during its lifetime. However, we can identify  
145 and reason about the fundamental classes of events that could arise during the system's  
146 lifetime and that are relevant to the system's security and dependability. Although these  
147 events are not exclusive to long-term operation, they become more likely as the designed  
148 lifetime of the system increases. We first summarize the main classes of challenges, and then  
149 discuss each in detail in the following subsections.

#### 150 **Subsystem failures**

151 In consequence of the above uncertainty about the root cause and nature of faults leading  
152 to subsystem failure, traditional solutions, such as enumerating all possible faults and  
153 devising mitigation strategies for each fault class individually, are no longer applicable.  
154 Instead, reasoning about these causes must anticipate a residual risk of unknown faults  
155 ultimately leading to subsystem failures and, treating them as faults of the system as a  
156 whole, mechanisms included that are capable of mitigating the effects of such failures at the

157 system level. Subsystem failures could be the result of random events or deliberate attacks,  
158 possibly due to new types of attack vectors being discovered. In the most general case, any  
159 type of subsystem could fail, including hardware failures, software attacks, and failures due  
160 to incorrect specifications of these subsystems. Subsystem failures may be hardware failures,  
161 software failures, subsystem compromises or specification failures. The open issues in these  
162 areas include:

163 **Hardware failures:** How can we protect spares (set our to take over in case of successful  
164 attacks or persistent faults) from environmental and adversarial influences, such that they  
165 remain available when they are required? How can we assert the sustainability of emerging  
166 material circuits, without at the same time giving adversaries the tools to stress and ultimately  
167 break these circuits? How can we protect confidential information in subsystems? How can  
168 we prevent one compromised hardware subsystem from compromising the integrity of others?  
169 How can we prevent adversaries from exploiting safety/security functionality of excluded  
170 components? How can we model erroneous hardware behavior? And, how can we construct  
171 inexpensive, fine grain isolation domains to confine such errors?

172 **Software failures:** How to design systems that can detect and isolate software subsystem  
173 failures? How to transfer software attack mitigation strategies between domains (e.g., PC to  
174 embedded)?

175 **Subsystem compromise:** How to recover a system when *multiple* subsystems are comprom-  
176 ised? How to detect subsystem compromised by a stealthy adversary? How to react to the  
177 detection of a (potentially) compromised subsystem? How to prevent the leakage of sensitive  
178 information from a compromised subsystem? How to securely re-provision a subsystem after  
179 all secrets have been leaked?

180 **Specification failures:** How to design subsystems that may fail at the implementation, but  
181 not at the specification level (and at what costs)? If specification faults are inevitable, how  
182 to design systems in which subsystems can follow different specifications whilst providing the  
183 same functionality, in order to benefit from diversity specifications and assumptions? How to  
184 recover when one of the essential systems has been broken due to a specification error (e.g.,  
185 how to recover from a compromised cryptographic subsystem)?

#### 186 **Requirement changes**

187 The requirements of the system could change during its lifetime. For example, the security  
188 requirements of a system could change due to new regulations (e.g., the EU General Data  
189 Protection Regulation) or shifts in societal expectations. The expected lifetime of the system  
190 itself could even be changed subsequent to its design. Requirement changes may be due to:

191 **Regulatory changes:** How to retroactively change the designed security, privacy, and/or  
192 safety guarantees of a system? How to prove that an already-deployed system complies with  
193 new regulations?

194 **User expectation changes:** How can a system be extended and adapted to meet new  
195 expectations after deployment? How to demonstrate to users and other stakeholders that an  
196 already-deployed system meets their new expectations?

## 4:6 Sustainable Security & Safety: Challenges and Opportunities

197 **Design lifetime changes:** How to determine whether a deployed system will retain its safety  
198 and security guarantees for an even longer lifetime? How to further extend the safety and  
199 security guarantees of a deployed system?

### 200 Environmental changes

201 The environment in which the system operates could change during the system’s design  
202 lifetime. This could include changes in other interdependent systems. For example, the  
203 United States government can selectively deny access to the GPS system, which is used by  
204 autonomous vehicles for localization.

205 **New threat vectors:** How to tolerate failure of subsystems due to unforeseeable threats?  
206 How to avoid single points of failure that could be susceptible to unforeseen threats? How  
207 to improve the modeling of couplings and dependencies between subsystems such that the  
208 space of “unforeseeable” threats can be minimized?

209 **Unilateral system/service changes:** How to design systems such that any third-party  
210 services on which they depend can be safely and securely changed? How can a system handle  
211 unilateral changes of (external) systems or services?

212 **Third-party system/service fails:** How to design systems such that any third-party services  
213 on which they depend can be safely and securely changed *after they have already failed*?  
214 How can a system handle the failure or unavailability of external services?

215 **Maintenance resource becomes unavailable:** How to identify all maintenance resources  
216 required by a system? How to maximize the *maintenance lifetime* of a system whilst  
217 minimizing cost? How to continue maintaining a system when a required resource becomes  
218 completely unavailable? How to ensure that a system remains secure and safe even under a  
219 new maintainer?

### 220 Maintainer changes

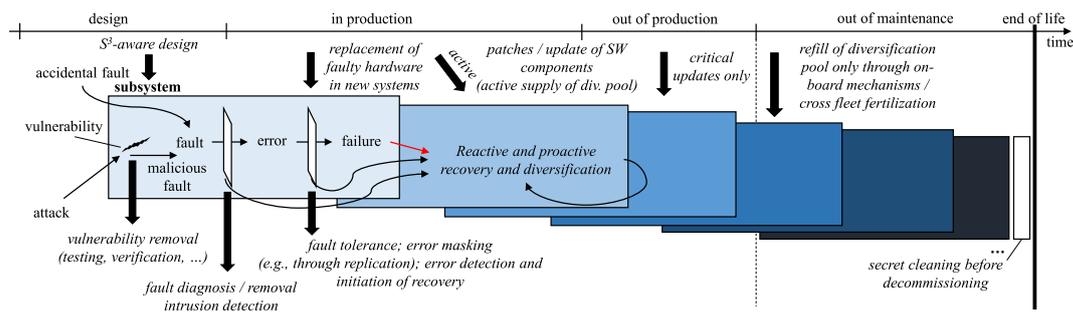
221 Most systems have the notion of a maintainer — the physical or logical entity that maintains  
222 the system for some part of its design lifetime. For example, in addition to the mechanical  
223 maintenance (replacing brakes, oil, etc.) Tesla vehicles regularly receive over-the-air software  
224 updates from the manufacturer.<sup>4</sup> For many systems, the maintainer needs to be trusted to  
225 sustain the security and safety of the maintained system. However, especially in long-lived  
226 systems, the maintainer may change (e.g., the vehicle is instead maintained by a third  
227 party service provider), which gives rise to both technical and non-technical challenges.  
228 Maintenance change requires answering:

229 **Implementing a change of maintainer:** How to securely inform the system that a change  
230 of maintainer has taken place?

231 **System becomes unmaintained:** How can a system decide that it is no longer maintained?  
232 How should an unmaintained system behave?

---

<sup>4</sup> <https://www.tesla.com/support/software-updates>



■ **Figure 2** Means to attain dependability and security.

### 233 Ownership changes

234 Finally, if a system has the notion of an owner, it is possible that this owner will change over  
 235 the lifetime of the system, especially if the system is long-lived. For example, vehicles are  
 236 often resold, perhaps even multiple times, during their design lifetimes. Change of ownership  
 237 has consequences because the owner usually has privileged access to the system. A system  
 238 may also be collectively owned (e.g., an apartment block may be collectively owned by the  
 239 owners of the individual apartments). Ownership may be:

240 **Cooperative:** How to securely inform the system about the change in ownership, without  
 241 opening a potential attack vector? How to erase sensitive data during transfer of ownership,  
 242 without allowing the previous owner to later erase usage/data of the new owner?

243 **Non-cooperative:** How to automatically detect non-cooperative ownership change? How  
 244 to erase sensitive data after loss of ownership, without allowing the previous owner to erase  
 245 usage/data of the new owner?

## 246 4 Technical Implications

247 The challenges identified in Section 3 lead to a number of technical implications, which we  
 248 highlight in the following before presenting more concrete proposals to address sustainable  
 249 security & safety in the next section. In particular, the realization that any subsystem  
 250 may fail, especially given currently unforeseeable threat vectors, and the realization that  
 251 subsystem failure is a fault in the overall system, demands a principled treatment of faults.

252 Although safety and security independently developed solutions to characterize the risk  
 253 associated with faults, we observe a gap to be closed, not only for automotive systems [11],  
 254 but also in any ICT system where risks are assessed based on the probability of occurrence.  
 255 Once exposed to adversaries, it is no longer sufficient to capture the likelihood of natural  
 256 causes coming together as probability distributions. Instead, probabilistic risk assessment  
 257 must take into consideration the incentives, luck, and intents of increasingly well-equipped  
 258 and highly skilled hacking teams. The conclusion may well be high risk probabilities, in  
 259 particular when irrational hackers (such as cyberterrorists) are taken into account. Also,  
 260 naively applying techniques, principles, and paradigms used in isolation in either safety or  
 261 security will not solve the whole problem, and worse, may interfere with each other. Moreover,  
 262 to reach sustainability, any such technique must withstand long-term operation, significantly  
 263 exceeding that of recent approaches to resilience.

264 For example, executing a service in a triple modular redundant fashion increases tolerance  
265 against accidental faults in one of the replicas. However, it does not help to protect a secret  
266 passed to the individual replicas by encrypting this secret for each replica and with the  
267 replica's key. Once the key of a single compromised replica is extracted, confidentiality is  
268 breached and the secret revealed. Further, if a replica fails due to an attack rather than a  
269 random fault, replaying the attack after resetting this replica will often recreate this failure  
270 and eventually exhaust the healthy majority.

271 To approach sustainable security & safety, we need a common descriptor for the root-cause  
272 of problems in both fields — *faults* — and a principled approach to address them [21, 20].  
273 As already pointed out in [4], faults can be accidental, for example, due to natural causes,  
274 following a stochastic distribution and hence justifying the argument brought forward in  
275 safety cases that the residual risk of a combination of rare events is within a threshold of  
276 accepted risks. However, faults can also be malicious, caused by intentional adversaries  
277 exploiting reachable vulnerabilities to penetrate the system and then having an up to 100%  
278 chance of triggering the combination of rare events that may lead to catastrophe. Faults  
279 can assume several facets, and the properties affected are the union of those both fields are  
280 concerned with: reliability, availability, maintainability, integrity, confidentiality, etc.

281 Returning to the above example, treating a confidentiality breach as a fault, it becomes  
282 evident why encrypting the whole secret for each replica individually constitutes a single  
283 point of failure. It also gives rise to possible solutions, such as secret sharing [16], but further  
284 research is required to ensure long-term secrecy for those application fields that require this,  
285 e.g., simple secret sharing schemes do not tolerate a mobile adversary that subsequently  
286 compromises and releases one replica at a time.

287 In the long run, failure of individual subsystems within the lifetime of the overall system  
288 becomes an expected occurrence. We therefore distinguish normal failure (e.g., ageing causing  
289 unavailability of a subsystem's functionality) from catastrophic failure (causing unintended  
290 system behavior), and devise means to attain dependability and security despite both.

## 291 4.1 S3 Lifecycle

292 Figure 2 sketches the path towards attaining dependability and security, and principled  
293 methods that can be applied over the lifetime of the system to obtain sustainable security &  
294 safety. Faults in subsystems manifest from attacks exploiting reachable vulnerabilities in  
295 the specification and implementation of the subsystem or from accidental causes. Without  
296 further provisioning, faults may manifest in errors, which may ultimately lead to failure  
297 of the subsystem. Here we take only the view of a single subsystem, which may well fail  
298 without inevitably implying system failure. More precisely, when extending the picture in  
299 Figure 2 to the overall system, subsystem failures manifest as system faults, which must be  
300 caught at the latest at the fault tolerance step. It may well be too late to recover the system  
301 as a whole after a failure has manifested, since this failure may already have compromised  
302 security or caused a catastrophe (indicated by the red arrow).

### 303 4.1.1 Design

304 Before deployment, the most important steps towards sustainable security & safety involve  
305 preparing the system as a whole for the consequences of long-term operation. In the next  
306 section, we sketch early insights what such a design may involve. Equally important is to  
307 reduce the number of vulnerabilities in the system, to raise the bar for adversaries. Fault  
308 and vulnerability prevention starts from a rigorous design and implementation, supported

309 for example by advanced testing techniques such as fuzzing [13, 14, 8, 9]. However, we  
310 also acknowledge the limitations of these approaches and consequently the infeasibility of  
311 zero-defect subsystems once they exceed a certain complexity, in particular in the light of  
312 unforeseeable threats. For this reason, intrusion detection and fault diagnosis and fault and  
313 vulnerability removal starts after the system goes into production,<sup>5</sup> which also is imperfect  
314 in detecting only a subset of intrusions and rarely those following unknown patterns while  
315 remaining within the operational perimeter of the subsystem. It is important to notice that  
316 despite the principled imperfection of fault, vulnerability and attack prevention, detection  
317 and removal techniques, they play an important role in increasing the time adversaries need  
318 to infiltrate the system and in assessing the current threat level, the system is exposed to.

### 319 4.1.2 In Production

320 While in production, replacement of faulty hardware components is still possible, by providing  
321 replacement parts to those systems that are already shipped and by not shipping new systems  
322 with known faulty parts. Software updates remain possible during the whole time when the  
323 system remains under maintenance, although development teams may already have been  
324 relocated to different projects after the production phase.

325 Fault tolerance, that is, a subsystem's ability to gracefully reduce its functionality to a  
326 non-harmful subset (e.g., by crashing if replicas start to deviate) or to mask errors (e.g., by  
327 outvoting the behaviour of compromised or malicious replicas behind a majority of healthy  
328 replicas, operating in consensus) forms the last line of defense before the subsystem failure  
329 manifests as a fault. The essential assumption for replication to catch errors not already  
330 captured by the fault and error removal stages is fault-independence of all replicas, which is  
331 also known as absence of common mode faults. Undetected common mode faults bear the  
332 risks of allowing compromise of all replicas simultaneously, which gives adversaries the ability  
333 to overpower the fault tolerance mechanisms. Crucial building blocks for replication-based  
334 fault tolerance are therefore the rejuvenation of replicas to repair already compromised  
335 replicas and in turn maintain over time the required threshold of healthy replicas (at least  
336 one for detecting and crashing and at least  $f + 1$  to mask up to  $f$  faults) and diversification  
337 to avoid replicas from failing simultaneously and to cancel adversarial knowledge how replicas  
338 can be compromised.

### 339 4.1.3 Out of Production

340 As long as the system is maintained, the rejuvenation and diversification infrastructure for  
341 the system may rely on an external supply of patches, updates and new, sufficiently diverse  
342 variants for the pool of diverse subsystem images.

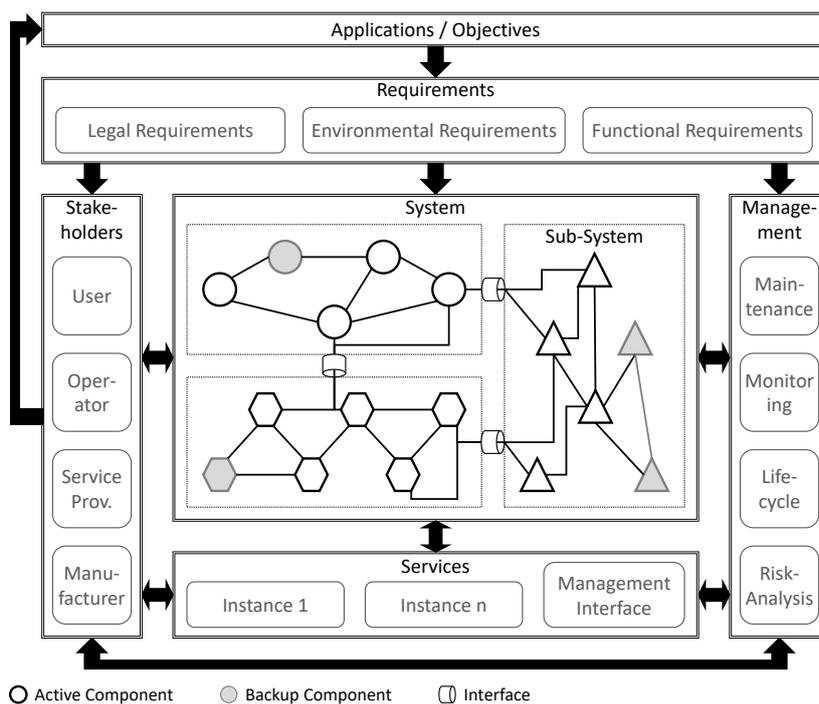
### 343 4.1.4 Out of Maintenance

344 Once the system falls out of active maintenance, replenishment of this pool is limited to  
345 on-board diversification mechanisms (such as binary obfuscation<sup>6</sup>) or through fleet-wide  
346 cross-fertilization, by exchanging diagnosis data between systems of the same kind, which  
347 allows one system to learn from the intrusions that happened to its peers (e.g., by mimicking

---

<sup>5</sup> We consider design-time penetration testing as part of the vulnerability removal process.

<sup>6</sup> [https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-sean\\_taylor-binary\\_obfuscation.pdf](https://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-sean_taylor-binary_obfuscation.pdf)



■ **Figure 3** Abstract view of a sustainable system and its surroundings.

348 gene crosscopying in patch generation for defense). The final state before the end of life of  
 349 the system is a reliable cleaning step of all secrets that remain until this time, followed by a  
 350 graceful shutdown of the system.

## 351 5 Design Principles

352 Figure 3 sketches one potential architecture for sustainable security & safety and shows the  
 353 abstract view of a sustainable systems and its connections/relations with its surroundings.  
 354 A system interacts (possibly in different phases of its life cycle) with different stakeholders.  
 355 The stakeholders (or a subset thereof) usually define the applications and objectives of a  
 356 systems, e.g., the manufacturer and developer define the primary/intended functionality of a  
 357 systems. The system's intended applications or objectives, as well as external factors such  
 358 legal frameworks, define the overall requirements a system must fulfill.

359 The center of Figure 3 shows the overall system that can be composed from multiple  
 360 subsystems, each of which is a combination of multiple components. Components of a  
 361 subsystem are connected with one another. Backup components (marked in grey) are  
 362 required to achieve fault tolerance, i.e., if one component of a subsystem fails it can be  
 363 (automatically) replaced by a backup component.<sup>7</sup>

364 Subsystems are connected and interact via defined interfaces (shown as tubes). As long  
 365 as the interfaces and the individual subsystems are fault tolerant, the overall system can  
 366 inherit this property.<sup>8</sup>

<sup>7</sup> Backup components can also be online/active to reduce the load per component as long as not failure has occurred.

<sup>8</sup> If an individual subsystem cannot provide the required fault tolerance level, the overall system requires

367 Sustainable systems usually do not operate in isolation. They often rely on external  
368 services, e.g., cloud services to execute computation intensive tasks. If these services are  
369 critical for the operation of the system, the resiliency of these services is relevant, e.g.,  
370 multiple instances of the service must be available, and the connection to the external service  
371 must be reliable, as well.

372 The system's management includes technical management as well as abstractly controlling  
373 the system and its operation. This has to cover both the management of the system itself  
374 and the external services upon which the system relies. The management can be performed  
375 by a single entity (stakeholder) or distributed among different stakeholders, e.g., the device  
376 manufacturer provides software updates for the system while the user configures and monitors  
377 it.

378 Based on the challenges discussed in the preceding section, we can already deduce certain  
379 architectural requirements and design principles for achieving S3:

380 ■ **Well-defined Components and Isolation**, limiting interaction to well-defined and  
381 constrained interfaces, to ensure complexity can be handled and to confine faults to  
382 causing components.

383 ■ **Avoid Single Points of Failure** For systems in long-term operation, it must be expected  
384 that any subsystem could fail, especially when exposed to unforeseen or unforeseeable  
385 attacks or changes. Anticipating this possibility, it is clear that no single subsystem can  
386 be responsible for the safety and security of the system as a whole.

387 ■ **Multiple Lines of Defense** Individual defenses can be overcome by adversaries, hence,  
388 relying on a single defense mechanism represents a single point of failure. Multiple lines  
389 of defense are needed to protect the system. Control-flow integrity (CFI) [1], for instance,  
390 has been attacked by full-function reuse attacks [6, 15] while randomization approaches  
391 suffer from information leakage [17]. Combining both can increase the security, e.g., by  
392 relying on randomization to mitigate the above attack to CFI.

393 ■ **Long-term Secrets and Confidentiality** Ensuring the confidentiality of data over long  
394 periods of time is very challenging since data, once leaked, cannot become confidential  
395 again. To prevent the leakage of secret information their use should be minimized.  
396 Additionally, single subsystem should not be allowed to access a secret as a whole, as this  
397 subsystem would become a single point of failure with respect to the secret information's  
398 confidentiality.

399 Let us give an example, detailed in [22]. Combining secret sharing, permanently re-  
400 encrypted data silos, function shipping into trusted execution environments, and sanitizing  
401 before revealing, it is possible to give access, even to bulk data, when it is required, but  
402 only in the form necessary. Rarely, applications require access to raw data. Instead, most  
403 just depend on aggregate, derived information which reveals much less of the secrets used.  
404 Revealing the classification of a deep neural network for a picture, while protecting the  
405 weights [19] is an example of such a secret protection scheme.

406 ■ **Robust Input Handling** (Sub)systems should not make assumptions with respect to  
407 the inputs it expects. This holds true for external as well as internal inputs. A robust  
408 system should incorporate: (a) it should cope well with noisy and uncertain real-world  
409 inputs, and (b) express its own knowledge and uncertainty of a proposed output despite  
410 unforeseen input [12].

---

redundancy with regard to that subsystem.

- 411 ■ **Contain Subsystem Failure** An immediate conclusion from the requirement to tolerate  
412 arbitrary failures is the need to confine each subsystem into an execution environment  
413 that acts as fault containment domain and in which the subsystem may be rejuvenated  
414 to re-establish the functionality it provides.
- 415 ■ **Replicate to Tolerate Subsystem Failure** If the functionality provided by the failing  
416 subsystem cannot be compensated by lower-level components (possibly at a different  
417 quality of service), the subsystem must be replicated to mask failure of individual replicas  
418 behind a majority of healthy replicas operating in consensus.
- 419 ■ **Diversify Nodes and Components** Replication is not sufficient to evade attacks [5];  
420 diversification is needed to avoid common mode faults and cancel adversary knowledge [18].
- 421 ■ **Adaption and reconfiguration** is needed to stay ahead of adversaries, but also to  
422 adjust to environmental changes.
- 423 ■ **Minimize Assumptions** Attackers may find new ways of violating assumptions to  
424 defeat safety or security measures that rely on them. Minimizing assumptions to those  
425 substantially required for the implementation, mitigates this threat.
- 426 ■ **Simplicity and Verifiability** While due to performance demands it is out of reach  
427 to keep the whole system simple, the trusted core typically required in fault-tolerant  
428 and secure architectures can be kept sufficiently small. Such a small and simple unit is  
429 furthermore less likely to suffer from unforeseen threats.

## 430 6 Conclusion

431 Achieving sustainable security & safety in real-world systems is a non-trivial challenge. It  
432 goes well beyond our current paradigms for building *secure* systems because it calls for  
433 consideration of safety aspects and anticipation of threats beyond the foreseeable threat  
434 horizon. It also goes beyond our current thinking about *safety* by broadening the scope  
435 to include deliberate faults induced by an adversary. Nevertheless, sustainable security &  
436 safety will become increasingly necessary as we become increasingly dependent on these  
437 (ICT) systems. Even if the full vision of sustainable security & safety is not fully achieved,  
438 any advances in this direction could have a significant impact on the design of future system.  
439 We have set out our vision for sustainable security & safety, identified the main challenges  
440 currently faced, and proposed a set of design principles towards overcoming these challenges  
441 and achieving this vision.

## 442 — References —

- 443 1 Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. Control-flow integrity principles,  
444 implementations, and applications. *ACM Trans. on Information System Security*, 13, 2009.
- 445 2 National Highway Traffic Safety Administration. Vehicle Survivability and Travel Mileage Sched-  
446 ules , 2006. URL: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/809952>.
- 447 3 Ferdinand Brasser Matthias Schunter N. Asokan Ahmad-Reza Sadeghi Paulo Esteves Verissimo  
448 Andreas Steininger Andrew Paverd, Marcus Völp and Thorsten Holz. Sustainable security  
449 & safety: Challenges and opportunities. long version avail. at: <http://www.icri-cars.org/wp-content/uploads/2019/01/s3-vision.pdf>, May 2019.
- 450 4 Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts  
451 and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and*  
452 *Secure Computing*, 1(1):11–33, Jan 2004. doi:10.1109/TDSC.2004.2.
- 453
- 454

- 455 5 Alysson Neves Bessani, Paulo Sousa, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo.  
456 The crucial way of critical infrastructure protection. *IEEE Security Privacy*, 6(6):44–51, Nov  
457 2008. doi:10.1109/MSP.2008.158.
- 458 6 Nicolas Carlini, Antonio Barresi, Mathias Payer, David Wagner, and Thomas R. Gross.  
459 Control-flow bending: On the effectiveness of control-flow integrity. In *24th USENIX Security  
460 Symposium*, USENIX Sec, 2015.
- 461 7 ENISA. Algorithms, key sizes and parameters report - 2013 recommendations. Technical  
462 report, [www.enisa.europa.eu](http://www.enisa.europa.eu), Oct. 2013.
- 463 8 Patrice Godefroid, Michael Y. Levin, and David Molnar. Automated whitebox fuzz testing.  
464 In *Annual Network & Distributed System Security Symposium (NDSS)*, 2008.
- 465 9 Patrice Godefroid, Michael Y. Levin, and David Molnar. Sage: Whitebox fuzzing for security  
466 testing. *Queue*, 10(1), January 2012.
- 467 10 ISO Technical Committee 22/SC 32. *ISO26262: Road vehicles - Functional safety*, 2018.
- 468 11 Antonio Lima, Francisco Rocha, Marcus Völp, and Paulo Esteves-Verissimo. Towards safe  
469 and secure autonomous and cooperative vehicle ecosystems. In *2nd ACM Workshop on Cyber-  
470 Physical Systems Security and Privacy (co-located with CCS)*, Vienna, Austria, October 2016.
- 471 12 Rowan McAllister, Yarin Gal, Alex Kendall, Mark Van Der Wilk, Amar Shah, Roberto  
472 Cipolla, and Adrian Weller. Concrete Problems for Autonomous Vehicle Safety: Advantages  
473 of Bayesian Deep Learning. In *Proceedings of the 26th International Joint Conference on  
474 Artificial Intelligence, IJCAI*, 2017.
- 475 13 Barton P. Miller, Louis Fredriksen, and Bryan So. An empirical study of the reliability of unix  
476 utilities. *Commun. ACM*, 33(12), December 1990.
- 477 14 Peter Oehlert. Violating Assumptions with Fuzzing. *IEEE S&P*, 3(2), March 2005.
- 478 15 Felix Schuster, Thomas Tendyck, Christopher Liebchen, Lucas Davi, Ahmad-Reza Sadeghi,  
479 and Thorsten Holz. Counterfeit object-oriented programming: On the difficulty of preventing  
480 code reuse attacks in C++ applications. In *36th IEEE Symp. on Security and Privacy*, 2015.
- 481 16 Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. URL:  
482 <http://doi.acm.org/10.1145/359168.359176>, doi:10.1145/359168.359176.
- 483 17 Kevin Snow, Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, Fabian Monrose,  
484 and Ahmad-Reza Sadeghi. Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained  
485 Address Space Layout Randomization. In *34th IEEE Symposium on Security and Privacy  
486 (Oakland 2013)*, 2013.
- 487 18 Paulo Sousa, Alysson Neves Bessani, Miguel Correia, Nuno Ferreira Neves, and Paulo Verissimo.  
488 Highly available intrusion-tolerant services with proactive-reactive recovery. *IEEE Transactions  
489 on Parallel and Distributed Systems*, 21(4):452–465, April 2010. doi:10.1109/TPDS.2009.83.
- 490 19 Shruti Tople, Karan Grover, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee.  
491 Privado: Practical and secure DNN inference. *CoRR*, abs/1810.00602, 2018. URL: <http://arxiv.org/abs/1810.00602>, arXiv:1810.00602.
- 492 20 Paulo Verissimo, Miguel Correia, Nuno Ferreira Neves, and Paulo Sousa. Intrusion-resilient  
493 middleware design and validation. In *Information Assurance, Security and Privacy Services*,  
494 volume 4 of *Handbooks in Information Systems*, pages 615–678. Emerald Group Publishing  
495 Limited, May 2009. URL: [http://www.navigators.di.fc.ul.pt/archive/papers/annals-  
496 IntTol-compacto.pdf](http://www.navigators.di.fc.ul.pt/archive/papers/annals-IntTol-compacto.pdf).
- 497 21 Paulo Verissimo, Nuno Ferreira Neves, and Miguel Correia. Intrusion-tolerant architectures:  
498 Concepts and design. In *Architecting Dependable Systems*, volume 2677 of *LNCS*, pages 3–36.  
499 Springer-Verlag, June 2003. Extended version in <http://hdl.handle.net/10455/2954>. URL:  
500 <http://www.navigators.di.fc.ul.pt/docs/abstracts/archit-03.html>.
- 501 22 Marcus Völp, Francisco Rocha, Jeremie Decouchant, Jiangshan Yu, and Paulo Esteves-  
502 Verissimo. Permanent reencryption: How to survive generations of cryptanalysts to come. In  
503 Frank Stajano, Jonathan Anderson, Bruce Christianson, and Vashek Matyáš, editors, *Security  
504 Protocols XXV*, pages 232–237, Cham, 2017. Springer International Publishing.
- 505